ATTR Syntax: Attr filename [permissions] Usage : Examine or change the security permissions of a file Opts: -perm = turn off specified permission perm= turn on specified permission -a = inhibit rms : d - directory file s - nc to owner w - write permit to own or - read permit to public pw - te permit to public BACKUP Syntax ge : Copies all data from one de ead error occurs s = single writes BASIC09 Syntax : Basic0 ge BUILD Syntax: Build filenar s from standard input CHD S nge working directory to specifi > Usage: Change execution directory to specified path CMP Syntax: Cmp filename1 filename2 Usage : File comparison utility COBBLER Syntax: Cobbler devname Usage : Creates OS-9 bootstrap file from current boot CONFIG Syntax: Config Usage : Create system boots and system disks COPY Syntax data from one fil E Syntax : Date [t Opts: t = specify ame> Usage : Check directory for wor sters -m = save of unused cluster nly -o = print <devname> {<devn : Del [-x] filenam s : -x = delete x: Deldir directo yntax: Dir [e] [x] the file names y x=print executi .] Usage : Display s converted characters to standard output DSAVE Syntax : Dsave [-opts] [dev] [pathname] Usage : Generates procedure file to copy all files in a directory system Opts : -b make a system disk by using OS9boot if present -b=<path> = make system disk using path as source -i = indent for directory levels -l = do not process b makdir command o num K ECHO Syn tandard output ED oriented text edito s text error messages for given error numbers EX Syntax: ex <modname> Usage: Chain to the given module FORMAT Syntax : Format <devname> Usage : Initializes an OS-9 diskette Opts ; R - Ready L - Logical format only "disk name" 1/2 number of sides 'No of

AUSTRALIAN OS9 NEWSLETTER

EDITOR :
Gordon Bentzen
8 Odin Street
SUNNYBANK Qld 4109

(07) 345-5141

SEPTEMBER 1989

AUSTRALIAN OS9 NEWSLETTER
Newsletter of the National OS9 User Group

EDITOR : Gordon Bentzen

HELPERS : Bob Devries and Don Berrie

SUPPORT : Brisbane OS9 Level 2 User Group.

Welcome to the 14th edition of the Australian OS9 Newsletter produced in sunny Queensland.
This edition represents the commencement of our second subscription year and the commitment to produce future editions through to August 1990. Just in case the above statements don't quite add up to you, we produced two newsletters as a trial in 1988, being July and August, and thanks to the support of the members we commenced a subscription period in September 1988.

We do hope that all members of the National OS9 User Group have enjoyed the many articles presented over the last year, and that we have all learnt something new about this powerful operating system.

At the time of writing, we have received a total of 25 subscription renewals which is more than the minimum we have set as being the basis for continuing this project. We are also confident that this number will increase.

This edition will be mailed to all members whose name has been in our database. However, the mailing list will be updated at END SEPTEMBER to include only financial members. So please send your renewal now so that you don't miss out on future editions.

I must praise the efforts of Bob Devries and Don Berrie and thank them both for their high quality articles contributed over the last year, because without their dedication to this project we would have had a lot of blank pages in each Newsletter.

Also a special thanks to members of the user group who have taken the trouble to send in articles and to share their knowledge with us. These include George Francis, Nickolas Marentes, "Rosko" McKay, Ian Clarke, Ross Pratt, and Phil Frost. Thanks for your participation.

As we have stated many times, this newsletter is intended as a medium by which we can all share in the combined knowledge of OS9 and its many application programmes, and it is only by many members being willing to pass on their experiences that this will be achieved.

We would like to present a question and answer series in the coming months to assist those members who are new to OS9 and those not so new to OS9. So come on, let us have those curly questions.

I have found from my own limited experience, that there is nothing like having to find the answer to a problem to make you learn. The only problem being that this can be very time consuming, and at times seem wasteful especially if someone already has the solution. We do not set ourselves up as experts by any means, but are willing to tackle any problem relating to the operating system. Having said that, I must also point out that the modification or debugging of commercial software packages is not our intent, as there does come a point where such a task is just not practical, or even possible, without the source code.

Well that is probably enough raving for any month, so let's move on to this month's edition and I trust that your continued support will result in some very interesting reading in the months ahead.

Until next month, Happy Computing. - Gordon Bentzen.

The following submission from Ole Eskildsen is presented as the first of a three part series. The content of this series serves as an example of just what can be done with OS9 (even with the humble Color Computer) and another reason why we all need to maintain interest in this versatile Operating System. Please let us have your comments so that we can endeavour to cater to your needs. Remember, no feedback can be very difficult to deal with.

Editor.

FOURTH GENERATION LANGUAGES (4GLs) AND OS-9

PART 1 - WHAT ARE 4 GLs AND WHAT IS AVAILABLE FOR OS-9?

This is the first part of what I think will become three parts (or perhaps more which may also depend on your reactions to the subject) on 4GLs, what they are, and what is available for the OS-9 operating system. At the end I hope to be able to make you a special offer to try out such a 4GL. For reasons of space availability in this newsletter I will obviously have to be fairly brief, however, I hope to cover the subjects sufficiently to benefit the general reader. The more experienced reader will probably find that in particular Part 1 does not provide him with a lot of new information, but please bear with me.

In the three parts I intend to cover the following subjects:

Part 1 - What Are 4GLs and What Is Available for OS-9?
Part 2 - A Real 4GL Called Sculptor
Part 3 - A Sample Application Using Sculptor and a Special Offer to Try It

So, what are 4GLs? As the name implies there must have been first, second and third generations before and perhaps fifth, sixth and seventh generations to follow. Let us therefore briefly review the history of computing as it relates to software.

Way back in the dim, dark ages of the forties the first computers started to appear and, as we all know now, the internal workings is based on the binary (two state) principle such as positive/negative, on/off and expressed mathematically as 0 and 1. In order to get the computer to work the "programmer" had to enter a long string represented by 0s and 1s which had the effect of switching the gates in the circuit on and off in order to finally arrive at the desired result. For instance, to add two numbers together he might have entered: "11110011001100000000000" This is machine language, the **FIRST GENERATION.** Little wonder you had to be a mathematics professor to program the early computers. Next someone decided to group the binary digits (bits) together in groups of four and use the hexadecimal numbering system to represent the bits so the string above becomes: "1111 0011 0011 0000 0000 0000" or hex "F 3 3 0 0 0" which incidentally in 6809 machine language means to add the value stored in hex address 3000 to the value in register D. This is somewhat easier to read, but to use it in the computer it was necessary to make a simple program that could translate the hex numbers into binary. Wow! Aren't you glad you don't have to program like that?

The **SECOND GENERATION** was born when someone decided that the above method was a bit too cumbersome for his liking and came on the idea of using mnemonics to represent the individual machine language instructions, such as add, subtract, store, move data, etc as well as performing jumps from one place in the program to another perhaps depending on a condition that had been tested for, and so on. This language is of course Assembler, where one mnemonic instruction correspond to one machine language instruction. This is therefore known as a 'low level' programming language because it is very close to the way the hardware (the Central Processing Unit or CPU) works and that also accounts for the fact that Assembler language differs from one type of processor to another such as Motorola 6809 (in the CoCo) and Intel 8088 in the original IBM PC. Still, this was a vast improvement over the first generation way of programming but again necessitated that a program was developed (in the first generation language) to

translate the programmer's Assembler language instructions and this program is simply known as "the Assembler". The add instruction shown above now looks like this: "ADDD $3000" in 6809 Assembler language which, I am sure you will agree, is a lot easier than the First Generation. Sometimes you hear a programmer say that he is programming in machine language. What he probably means is that he is programming in Assembler language and his program is assembled into machine language. As you can probably imagine this is the most effecient way of programming and that is the reason why most system functions and system software is written in Assembler although there is at least one notable exception which I will explain under Third Generation languages.

The **THIRD GENERATION** languages came into existence when some programmers realized that it was too tedious to program in Assembler language since even a short program may soon run into hundreds and thousands of instructions. The idea was therefore conceived to develop a 'high level' programming language where one instruction may be translated into many machine language instructions. In other words, it would now be possible to write programs with fewer instructions which would mean that it could be written and debugged in a shorter time (fewer instructions = fewer potential bugs). Two of the first 3GLs were FORTRAN and COBOL. Other 3GLs include (yes, you guessed it) BASIC as well as PASCAL, RPG, PL1, and many others. How would you now perform the example instruction shown in the preceeding paragraphs? Well, you cannot! (At least I don't think you can in any of the languages mentioned.) This is why 3GLs are called 'high level' languages. I said in the previous paragraph that there was an exception and that is the 'C' programming language, which is a very powerful 3GL but at the same time it allows the programmer to dive down to a low level close to the hardware whenever required and as a result C is also very useful for systems programming. Of course the 3GL cannot directly be executed by the processor, so a translating program, called a compiler, had to be developed for each language. This introduced a new benefit, namely that of portability of an application. By developing compilers for e.g. COBOL in Assembler language for many different processors it became possible to write a program in COBOL and then port the source program to different hardware and recompile it using the resident COBOL compiler.

Now then, what about **FOURTH GENERATION** languages? Well, there seems to be several different opinions about what constitutes a 4GL, however, we can probably agree that it has to be even more powerful than a 3GL, i.e. one 4GL instruction must result in even more work being done by the processor. All of this is aimed at improving the productivity of the programmer so that he can churn out more and more solutions in as short a timeframe as possible. 4GLs usually have some of the following characteristics. 1) A powerful database management system (DBMS), perhaps a relational DBMS, which will store and retrieve the data for the programmer without him having to actually tell the system where or how to do this, thus removing an enormous burden from the progrrammer. 2) A powerful programming language which performs the most work with the least number of instructions. 3) A program generator of some sort whereby the programmer or perhaps even the end user can specify his requirements and the generator then automatically 'writes' the program (with no bugs). 4) The ability to modify (or fine tune) a program generated by the program generator. It should be easier to learn than 3GLs and it should be possible for an experienced programmer to produce a desired result in a much shorter timeframe, some say as much as ten times faster than using a 3GL, this however, would vary from language to language, still it would be a considerable improvement.

One such 4GL is SCULPTOR which I have used on and off for the last couple of years to develop real applications in that are installed and running at customer sites. In Part 2 I will describe some of the main functions of Sculptor and in Part 3 I will present a sample application. The Sculptor distributor for Australia has kindly agreed to provide a demo version so that you can try out the sample application. If you like it you can then obtain Sculptor either from myself or from the distributor. There may be other 4GLs available for OS-9 but I have never seen them, but I would of course be interested to hear about and particularly to try out any other 4GLs. Questions or comments should be addressed to:

Ole Eskildsen
11 Monarch Street
Kingston QLD 4114
Tel: (07) 209 4322

OS-9 PROFILE
A review By. Gordon Bentzen

One of the very first practical uses of a computer usually thought of, is to organize lists of data of some kind. The requirements can vary from a simple file of Name, Address etc. to a rather complex list of variables. Bob Devries has featured his 'C' database in this newsletter which is a good example of some of the complexities in creating a simple name and address type database.

Many commercial database programmes have been developed for most operating systems, and of these, opinion as to the best will vary widely. "Sculptor" for the OS-9 operating system is probably the most powerful database for the CoCo and perhaps could be comparable to many powerful databases available for P.C's running other operating systems.

The purpose of this article however, is not to present views on all the available database programmes available, but to comment on just one of these, i.e. PROFILE. The Computerware "Profile" is available in RS-DOS and OS-9 versions which from a users point of view are very similar. The OS-9 version distributed by Tandy for the CoCo will run under OS-9 level 1 only and will format its displays to the available screens of the system. Profile will present a correctly formatted display in the standard 32 column screen and will also give the same pleasing results if the OS-9 system uses an 80 column display produced by "WordPak". OS-9 Level 2 users should not stop reading here as I will present a couple of patches to Profile for Level 2 systems.

Profile is simple to set up and use, and has many features which will allow the management of information in such a variety of ways that it should suit most requirements. Profile is Menu driven by interactive menus which makes it extremely easy to use once all the database structures have been defined.

The first step of course, is to define the "Record" format. To do this the type and size of each field must be decided. When creating a new database, simply type in a name for the database in the opening menu. Profile will look in the current data directory for that name and when not found a message will advise you of this fact and give you a y/n option to create a database of that name.

Field types available are :-
Type 1 = Alphanumeric
Type 2 = Date (default format of mm/dd/yy may limit its suitability)
Type 3 = Math (used in the four normal math functions plus report totals)
Type 4 = Derived (calculated by Profile)

Limitations: A record may contain a maximum of 35 fields.
Fields may contain up to 64 characters.
The maximum capacity for any one record is 512 characters.

Features: Up to 9 different screen display formats may be defined for each database file.
Up to 9 different file access methods may be defined (sort methods).
Up to 9 different report formats may be defined.
Up to 3 different levels of sorting per method.
Read "Dynacalc" files into the database.
Save database files in "Dynacalc" format.
Restore deleted records.
Compress database files (this effectively removes deleted records- deleted records not restorable).
Each report format may include one (1) set of printer codes. e.g. code for compressed print.

Now some of the "bugs" or unexpected results, and I have not found many.
When a record is simply deleted, it is not accessible in screen displays as one would expect, however the deleted record WILL be included in a report when the key field (access method) is different to the key field under which the deletion was made. The only sure way of excluding deleted records from reports is to compress the database file prior to running reports. Perhaps this bug is a predictable result when you think about it, but it has caught me more than once.

Reports offer the option of entering a 'selection criteria' at report run time, which can be very useful. The selection criteria may include all the usual logical operators of =, <, >, which are not case sensitive in alpha fields. The selection criteria must be based on the current key field (the current access method's primary field).

I have noted that Profile stores all the defined screen, report and other format files, in fact all files related to a database, under a directory. The directory name is the name given at creation of the database. Profile will automatically create this directory and sets the 'pe' (public execute) attribute off. This allows Profile to distinguish its own database directories from other normal directories which may be on the same disk.

While this may not be a foolproof method, it does seem to work as desired most of the time. The database may of course be copied under OS-9 in the normal way of makdir etc. If you use this method and want Profile to read the directory as the name of a database file, you will need to set the pe attribute off, e.g. attr filename -pe

Conclusions: OS-9 Profile is very flexible and easy to use and will allow a database to be set up with only a little practice. The limitation of 512 characters in any one record may be a problem for a complex database, however I have found that all my applications on the CoCo are comfortably accommodated within this limit.

So, if you do not have a copy of OS-9 Profile, I suggest a phone around of your local Tandy stores and you may be lucky enough to pick up a copy. Since Intertan Australia have dropped the CoCo in Australia we have noted many bargains at clearance prices for CoCo hardware and software though you may have to resort to a purchase direct from the U.S. On the other hand, you may have already picked up a copy at the bargain prices and now are a little disappointed that it won't run under OS-9 Level 2. The following patches will have you smiling again.

PATCH OS9 PROFILE FOR LEVEL 2

OS9 Profile consists of two programme modules, PROFILE, the main database, and MGT the database management utilities. Both of these programme modules look for the OS9 system module SYSGO, which of course is a Level 1 module. Simply changing the code to look for the CC3GO Level 2 module will not work, so the answer is to change the code so that the programme does not look for SYSGO.

The easiest way to patch Profile and MGT is to create a modpatch script using 'Build' 'Edit' or any text processor of your choice. Then load the module to be patched (e.g. Profile) into memory, and use 'Modpatch' with the script file as a command line parameter. e.g. modpatch patch_profile

Patch_PROFILE (This is the name under which to create the modpatch script)

```
L PROFILE
C 09 11 12
C 0A E4 27
C 1227 81 32
C 1228 59 C9
C 1229 26 04
C 122A F5 13
C 122B 35 9F
C 122C 70 09
V
```

(The modpatch script file should contain the above boldtype lines and be located in the current data directory)

Now, **load /d1/cmds/profile** (this line assumes that the Original 'Profile' is loaded from /d1)
modpatch patch_profile
save /d0/cmds/profile profile

That is all there is to it. The modules 'load' 'modpatch' and 'save' must be either in memory or the current execution directory (/d0/cmds in the example above). 'Save' comes with the Level 2 Development System, or from OS9

Level 1 (they are identical)

The procedure for patching MGT will require similar commands. 'MGT' should be substituted for each occurrence of 'profile'. Here is the modpatch script to patch MGT.

Patch_MGT

```
L MGT
C 0A 00 43
C 543 81 32
C 544 59 C9
C 545 26 04
C 546 F5 13
C 547 35 9F
C 548 70 09
V
```

Now, load /d1/cmds/mgt
modpatch patch_mgt
save /d0/cmds/mgt

oooooooooo0000000000oooooooooo

A Database in C
By Bob Devries

Included in this month's section of the code for the database is the functions insert() and usage(), which, as their name suggests, do the following tasks. Insert takes a data from the keyboard and stores it into the database record structure, and when everything is correct, it calls the save() function to save it to the disk. The usage() function, only used when a command line error is encountered, tells the user what he did wrong, and quits the programme.

Next month will be the last installment of the database, in which I will be presenting the match() and amend() functions, and also give you some details on compiling the full code. Of course, any errors that crop up will be addressed as space permits, and when you let me know that something went wrong. I hope you will have fun from playing with this little (!) programme. By the way, in case I had not mentioned it before, each section of the code, can be compiled by itself, by forcing the compiler to include the header file, and stopping before the link stage with the '-r' switch. Well, without furtner ado, here's this month's piece of code.

```
insert()
/* this routine inputs data into the mail structure and keeps repeating */
/* until the user is satisfied it is correct, and then saves it. */
/* it returns a long value for the record number and requires no params */
/* strncpy() was used to input the strings to stop overflow of fields */
(
        char ch;
        char tempstr[21];       /* temporary string for data input */
        int temprec = 1;        /* temporary record number */
        int replace = FALSE;    /* flag to signal write to end of file */

        while (TRUE)
                {
                scrnmask();     /* display empty screen mask */
                cursor(14,5);
                gets(tempstr);
                strncpy(mail.surname,tempstr,20);
                cursor(51,5);
```

```
                gets(tempstr);
                strncpy(mail.firstname,tempstr,20);
                cursor(13,7);
                gets(tempstr);
                strncpy(mail.street,tempstr,20);
                cursor(11,8);
                gets(tempstr);
                strncpy(mail.city,tempstr,20);
                cursor(12,9);
                gets(tempstr);
                strncpy(mail.state,tempstr,3);
                cursor(15,10);
                gets(tempstr);
                mail.postcode = atoi(tempstr);
                cursor(10,23);
                eraseln();
                cursor(10,23);
                printf("All correct ? y or n ");       /* display prompt */

                ch = '\0';
                while ((ch != 'N') && (ch != 'Y'))
                        ch = toupper(getch());
                cursor(10,23);
                eraseln();
                if (ch == 'Y')          /* if no , go do it again */
                break;
        }
        if (mail.surname[0] == '\0')    /* if no valid input quit */
                return(temprec);        /* without saving record  */
        temprec = save(temprec,replace);        /* save record and */
        return(temprec);                        /* return record number */
}

usage()         /* usage function in case of incorrect comm line */
{
        printf("\n\n        Usage : database <dbfile>.\n\n");
}
```

oooooooooo0000000000ooooooooo

CALENDAR PROGRAM

By Ole Eskildsen

Here is a little quicky in BASIC09. This program will print a calendar to the standard output (the screen). It is a program I have converted from a Microsoft basic program which I keyed in years ago on a Tandy Model III (no not a CoCo-3) from somewhere I cannot now remember. In the conversion to BASIC09 I didn't have to do very much, but one of the things I did which is not actually necessary was to remove all the unnessary line numbers to make the program more readable in BASIC09. What I am telling you is that many programs that run in Microsoft BASIC can be converted to BASIC09 with very little and sometimes nil effort. Maybe someone should make that a subject of another article.

To use Cal, simply place it in your execution directory together with RunB and then type :

```
cal 8 1989
```

and you should see the month of August 1989 on your screen. This is provided you use the Shell called ShellPlus

available from the User Group public domain library. If you use the standard Shell from Tandy then...

type: cal ("8","1989")

Not very neat, don't you agree?

The syntax for using Cal is almost the same as the UNIX Cal function, but one of the differences is that if you want to get the whole year then...

type: cal 0 1989

If you want to print the calendar on your printer simply...

type: cal 8 1989 > /p

You could also redirect it to a file thusly...

type: cal 0 1989 > /d1/calendar1989

and then use your editor or word processor to add a message, print it out and send it to your friends at the end of the year.

Please enjoy it and happy computing... Cheers, Ole

```
PROCEDURE Cal
0000      (* Calendar program similar to UNIX Cal
0027      (* By: Ole Eskildsen, 1989
0041      (* Placed in the Public Domain and it may freely be copied and used
0084      (* for non-commercial purposes.
00A3
00A4      (* Usage: Cal month year
00BC      (* if month = 0 then calendar for whole year is printed.
00F4      (* Note: month parameter is mandatory, unlike the UNIX cousin.
0132
0133      PARAM mth,yr:STRING[4]
0143
0144      DIM month,year:REAL
014F      DIM A,B,C,D,P,R:INTEGER
016A      DIM J:REAL
0171
0172      month=VAL(mth)
0178      year=VAL(yr)
0184
0185 200  IF month<0 OR month>12 OR year<1600 OR year>2399 THEN
01AF        PRINT "Usage: Cal month year"
01C8        PRINT "       Displays calendar, may be redirected."
01F8        PRINT "       if month=0 entire year displayed"
0223        PRINT "       Valid year range: 1600-2399"
0249        PRINT
024B        END
024D      ENDIF
024F
0250      A=0 \B=0 \C=0 \D=0 \J=0 \P=0 \R=0
0282      PRINT
0284
0285      C=6
```

```
028C      FOR J=1600 TO year
029F        IF J=year THEN 3210
02AF        IF J/4<>INT(J/4) THEN 3200
02C8        IF (J-1700)*(J-1800)*(J-1900)*(J-2100)*(J-2200)*(J-2300)=
      0 THEN 3200
030A        C=C+2
0315        GOTO 3210
0319 3200   C=C+1
0327 3210   IF C<7 THEN 3230
0339        C=C-7
0344 3230 NEXT J
0352
0353      PRINT  \ PRINT
0357
0358 3250 FOR R=1 TO 12
036B        READ A$
0370        READ B
0375        IF year/4<>INT(year/4) THEN 3320
038E        IF A$<>"FEBRUARY" THEN 3320
03A5        B=B+1
03B0
03B1        (* Print calendar
03C2 3320   IF month=0 OR month=R THEN
03DB          PRINT TAB(14); A$; " "; year
03EC          PRINT "========================================="
0419          PRINT " SUN   MON   TUE   WED   THU   FRI   SAT "
0446          PRINT "========================================="
0473          FOR D=1 TO B
0484            PRINT TAB(6*C+2);  \ PRINT USING "I3>"; D;  \
049F            C=C+1
04AA            IF C<7 THEN 3420
04B9            PRINT " " \
04BF            C=0
04C6 3420     NEXT D
04D4          PRINT
04D6          PRINT "========================================="
0503 3450     FOR P=1 TO 2
0516            PRINT
0518 3470     NEXT P
0526        ELSE
052A          FOR D=1 TO B
053B            C=C+1
0546            IF C=7 THEN
0552              C=0
0559            ENDIF
055B          NEXT D
0566        ENDIF
0568 3480 NEXT R
0576 3490 DATA "JANUARY",31,"FEBRUARY",28,"MARCH",31,"APRIL",30
05AE      DATA "MAY",31,"JUNE",30,"JULY",31,"AUGUST",31,"SEPTEMBER"
05E7      DATA 30,"OCTOBER",31,"NOVEMBER",30,"DECEMBER",31
0617
0618
0619
```